DOCUMENT RESUME

ED 242 294 IR 011 003

AUTHOR Garraway, R. W. T.
TITLE Microcomputer Based Computer-Assisted Learning System: CASTLE.
PUB DATE Oct 83
NOTE 9p.; Paper Prepared for the Canadian Symposium on Instructional Technology (4th Winnipeg, Canada, October 19-21, 1983).
PUB TYPE Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Authoring Aids (Programing); *Computer Assisted Instruction; *Computer Software; *Design Requirements; Instructional Development; *Microcomputers; *Program Development; Program Evaluation; Programing; Programing Languages; Systems Development
IDENTIFIERS NATAL Programing Language

ABSTRACT

The purpose of this study was to investigate the extent to which a sophisticated computer assisted instruction (CAI) system could be implemented on the type of microcomputer system currently found in the schools. A method was devised for comparing CAI languages and was used to rank five common CAI languages. The highest ranked language, NATAL, Canada's National Authoring Language, was used as the basis for the design of a new, microcomputer-based CAI system named CASTLE (Computer Assisted Student Tutorial Learning Environment). Although primarily designed for trained CAI authors and researchers, the system has features that could assist the beginning author in creating CAI lessons and courses. The design specifications for the CASTLE language and the CASTLE support system were defined and a subset of the language and support system was developed on a Commodore CBM 8096 microcomputer system. The CASTLE system software was written in COMAL-80, and techniques of incremental compilation in an interactive and interrogative environment were used. The completed system was evaluated and recommendations made for further research and development. Two references are listed. Abstracts are provided in English and French. (Author/LMM)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED242294

MICROCOMPUTER BASED COMPUTER-ASSISTED LEARNING SYSTEM: CASTLE

R.W.T. Garraway

Div. of Educational Research Services

The University of Alberta

Edmonton, Alberta

Paper prepared for the
Fourth Canadian Symposium on Instructional Technology
Winnipeg, Canada, October 19-21, 1983

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

R.W.T. Garraway

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

IR011003

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

✓ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

Session 19 - M
Oct. 21, 09:30

MICROCOMPUTER BASED COMPUTER-ASSISTED LEARNING SYSTEM: CASTLE

R.W.T. Garraway

Div. of Educational Research Services
The University of Alberta
Edmonton, Alberta

The purpose of this study was to investigate the extent to which a sophisticated CAI/CAL system could be implemented on a typical microcomputer system currently found in the schools. A method for comparing CAL languages was devised and used to rank five common CAL languages. NATAL, Canada's National Authoring Language, was found to have the highest rank and so was used as the basis for the design of a microcomputer based CAL system.

The new system, named CASTLE (Computer Assisted Student Tutorial Learning Environment), was primarily designed for trained CAL authors and researchers, but has features that could assist the beginning author in creating CAL lessons and courses. The design specifications for the CASTLE language and the CASTLE support System were defined.

A sub-set of the CASTLE language and support system was developed on a Commodore CBM 8096 microcomputer system. The CASTLE system software was written in COMAL-80. Techniques of incremental compilation in an interactive and interrogative environment were used. The completed system was evaluated and recommendations made for further research and development.

CASTLE - SYSTÈME D'ENSEIGNEMENT AUTOMATISÉ SUR MICRO-ORDINATEUR

Notre projet avait pour but d'étudier dans quelle mesure un système élaboré d'enseignement automatisé peut être exploité grâce au micro-ordinateur que l'on trouve couramment dans nos écoles. On a élaboré une méthode permettant de comparer divers langages d'enseignement automatisé et l'on a ainsi évalué cinq langages courants. NATAL, le langage de rédaction national du Canada, a obtenu la meilleure cote et a donc servi de base à l'élaboration d'un système d'enseignement automatisé sur micro-ordinateur.

Le nouveau système, appelé CASTLE (soit Computer Assisted Student Tutorial Learning Environment - système d'enseignement automatisé élève-tuteur), a d'abord été conçu en fonction des auteurs et des chercheurs habitués à l'enseignement automatisé, mais il comporte néanmoins certaines fonctions visant à assister le nouvel auteur qui veut créer des leçons et des cours automatisés. On a donc défini les caractéristiques de conception du langage CASTLE ainsi qu'un système de soutien.

On a également mis au point sur un système à micro-ordinateur Commodore CBM 8096 un sous-ensemble de langage et de système de soutien CASTLE. Le logiciel du système a été rédigé en COMAL-80. On a recouru à des techniques de compilation par incrémentation dans un contexte de dialogue et d'interrogation. On a ensuite évalué l'ensemble du système, une fois qu'il a été achevé et s'on a fait des recommandations quant à la recherche et au développement futurs.

IR011003

Paper prepared for the
Fourth Canadian Symposium on Instructional Technology
Winnipeg, Canada October 19-21, 1983

MICROCOMPUTER BASED COMPUTER-ASSISTED LEARNING SYSTEM: CASTLE

R.W.T. Garraway

The Division of Educational Research Services
The Faculty of Education
The University of Alberta

I. Introduction

> The debate over whether or not microcomputers will substantially impact education has all but vanished. Tens of thousands of micros have made their way into the schools, and discussions have at last turned to the planning needs ... the preparation of software and courseware (software which teaches) for use with microcomputers.
> (Roblyer, 1981, p. 47)

With this influx of microcomputers into the schools there has been a resurgence of interest among teachers and school administrators in computer-assisted instruction (CAI). Most have been scanning the market place for good courseware for use with their students. Many have expressed an interest in writing their own CAI programs. Unfortunately good courseware for microcomputers is rare and specialized CAI software systems almost non-existent or of limited sophistication.

A. The Development of a Microcomputer Based CAI System

The only CAI language widely supported on todays microcomputers is PILOT. It is relatively easy to learn and use, but does not support modern structured programming techniques, built-in performance recording or student restart points, and has only one type of answer analysis. Various courseware development systems designed for use by inexperienced authors have appeared on the market. Many of these are useful for implementing simplistic CAI but they lack many of the features considered 'standard' on main-frame CAI systems. Even the expensive PASS system developed by Bell & Howell, though having many fine features, lacks expansibility, locking the experienced author into set instructional strategies.

The purpose of the CASTLE project was to investigate the extent to which a sophisticated CAI system could be implemented on a typical microcomputer system currently found in the schools. This system should have features useful to researchers in CAI. These might include flexible specification by the author of the information to be included in performance records; and author developed response comparison functions, input edit functions, and general functions.

B. A Comparison of Five CAI Languages

Voyce (1979), during his research to develop a multilingual CAI system, proposed a taxonomy of functional properties of CAI languages. (p. 66) Voyce's taxonomy was used to develop a system for comparing CAI languages. The taxonomy of functional properties was divided into eighteen domains termed CAI language features. Within each domain the functional properties are considered compensatory. That is, a weakness in one functional property may be compensated for by a strength in another. To compare a set of languages a value is given to each property in a domain that differentiates the languages. The value is in the range of one to three which represents that property's compensatory weight within the domain. When assessing a particular property in one of the selected languages, a value of zero is assigned if the property is not present; or an integer value, up to the maximum set for that property, is assigned which depends on the extent to which that property is implemented in the language.

Once the values for each of the functional properties for all of the languages have been assessed, they are summed for each language within each CAI language feature domain. Thus each language has eighteen scores, one for each feature domain. These scores are used to obtain a profile index in the range zero to ten. This profile index is called a relative prominence index. It is obtained for each feature by dividing a language's score on that feature by the score of the language having the highest score on that feature and multiplying the result by ten.

Using the above scheme, five CAI Languages were compared: CAN-7, COURSEWRITER III, NATAL, TUTOR and PILOT. Scores for each of the eighteen language feature domains for all five languages were assessed. These are shown in Table 1. This table also contains a key to the abbreviations used for the names of the CAI languages. From these scores the relauve prominence indices were derived.

Table 1 CAI Language Features: Raw Scores

	N*	T	C	CW	P
1. Data					
1.1 Variables	22	13	4	4	6
1.2 Types and Operations	40	27	22	9	29
1.3 Pattern Matching	28	23	14	11	10
1.4 Student's Name, Date, Time	9	4	2	2	0
2. Data Structures					
2.1 Organization and Operations	15	11	10	7	6
2.2 External Storage	11	3	6	2	5
3. Data Conversion	13	8	6	3	7
4. Program Control					
4.1 Labels	6	3	5	6	4
4.2 Transfer of Control	3	2	3	1	0
4.3 Block Instructions	7	5	0	0	0
4.4 Subprograms	20	11	10	9	6
4.5 Stop/Start Student Sessions	3	2	2	3	0
4.6 Implicit Activity	6	5	2	4	1
5. Output to Student					
5.1 Student Output Devices	6	6	7	7	1
5.2 Terminal Display	11	9	5	1	7
5.3 The Display Area	27	16	7	4	9
5.4 Additional Display Functions	12	12	4	7	5
6. Student Response	10	9	4	10	6

*Abbreviations: N - NATAL, T - TUTOR, C - CAN-7
CW - COURSEWRITER III, P - PILOT

NATAL had the highest score on all features but one. Table 2 lists the number of times each language obtained a higher feature score over each of the other languages. Though each feature is not of equal importance in a CAI language, the table does indicate a ranking of the languages compared. NATAL has the highest rank followed by TUTOR, CAN-7, COURSEWRITER III and PILOT, in that order.

Figure 1 provides a profile of relative prominence indices for each langage over the eighteen CAI language features. They are displayed on three graphs for clarity.

IR011003

Table 2 Frequency of Obtaining Higher Feature Score

of/over	N	T	C	CW	P	TOTAL
NATAL		16	16	14	18	64
TUTOR	0		13	14	15	42
CAN-7	1	4		9	10	24
CWIII	1	4	5		11	21
PILOT	0	3	7	6		16

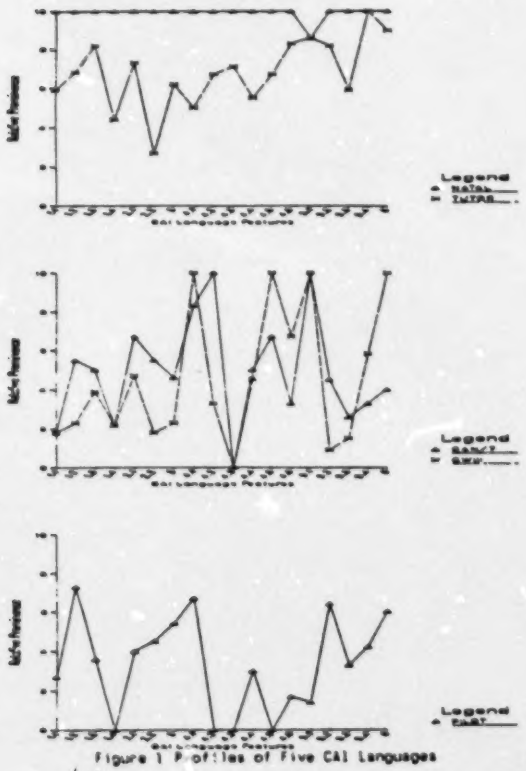

Figure 1 Profiles of Five CAI Languages

II. Design Specifications for the CASTLE Language

A. Overview

NATAL is too large and complex a language to be easily implemented on the microcomputers most often found in schools today. However, the design philosophy of NATAL, separating strategy and content, could be incorporated into an extended version of a language already available on these same microcomputers. That was the approach taken in this project. The name selected for this CAI development system is CASTLE, an acronym for *C*omputer-*A*ssisted *S*tudent *T*utorial *L*earning *E*nvironment. In fact, CASTLE was developed as a NATAL 'look-alike', having a procedure language, an instructional unit language, and a display sub-language.

A CASTLE course is composed of one or more lessons. The order of lessons in a course is, primarily, under the control of the author. The author defines a default order for lessons within a course, but from within any lesson, there may be any number of exits to other lessons. Thus the author may individualize each student's pathway through a course (see Figure 2).

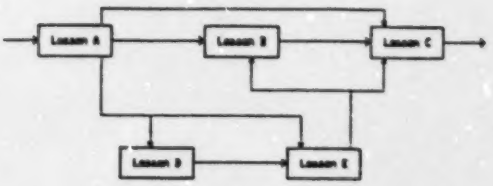

Figure 2 Illustrative Example of Lesson Selection Within a Chapter

The lesson is the basic control module in a CASTLE course. It must have a unique name within a course. Lessons are written in the CASTLE procedure language. Each lesson begins with a lesson entry procedure named 'lesson'. A procedure is a computer routine, or sub-program, that controls the actions of the computer. In CASTLE, procedures are used to select instructional units, gather data on computer interactions with the student, and make decisions on choosing the student's pathway through a course.

The author may divide a lesson into parts. A part might be associated with a single objective or sub-objective of a lesson. Upon re-entering a course the student is automatically routed to the beginning of that part of the lesson which was being executed just prior to sign off. Each part of a lesson has a unique number and name within the lesson, and begins with a part entry procedure having the part's name. Parts would normally be executed in numerical order but the author has complete freedom to call parts in any order. For example, a student having difficulty in part 4 may be re-directed by the author to part 2.

Within lessons, procedures may be defined by the author for the general control of the lesson. A procedure may be called by another procedure and is then subordinate to the calling procedure (see Figure 3).

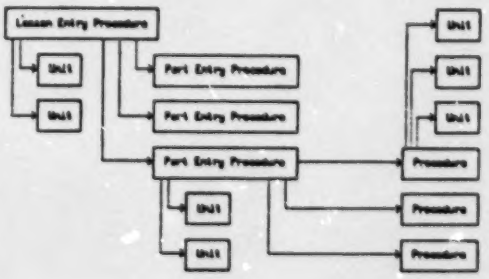

Figure 3 Procedures and Units in a Lesson

Instructional transactions are carried out by unit routines. All interaction with the student is handled by these units. The unit is used to present information to the student, to pose questions and to analyze the student's response. System variables are used to pass information about the transaction back to the calling routine. Units are called from procedures in the lesson modules. The presentation of information is handled by a special display sub-language that formats text and graphics on the student's display devices in a manner similar to a word processor. The unit language and display sub-language are virtually the same as their NATAL counterparts.

Routines that support both instructional transactions and control procedures are called functions. These are classed as general functions, edit functions, comparison functions, and graphics functions. The CASTLE system provides a number of functions in all four classes but the author may define unique function's in each class using the CASTLE procedure language (see Figure 4).

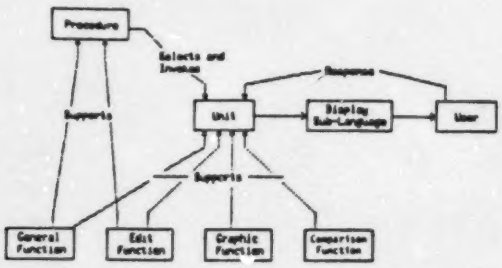

Figure 4 Unit, Procedure, and Function Relationships

The CASTLE Language is composed of three separate language systems that share system data. The system data are of three classes: system files, system registers, and system variables. The language systems are the procedure language, the instructional unit language, and the display sub-language.

B. System Data

System Files

There are five types of system files: lesson module files, unit parse table files, unit string table files, display files, and window files. The lesson module file type contains CASTLE system routines and author defined procedures and functions. The two unit file types contain control and display data for the unit language interpreter and the display sub-language interpreter. The remaining two file types are referenced by the display sub-language interpreter. Display files contain data to be formatted by the display sub-language interpreter. Window files are preformatted data to be displayed in an author-defined window on the student's display screen. These files are referenced by file identifiers which may have up to fifteen alphanumeric and/or symbol characters except the quotation mark, semicolon, asterisk, question mark and comma, but may include the space character. The first character must be alphanumeric. A file identifier is known to the entire CASTLE system.

Unit and lesson identifiers may also have numeric indices. They appear after the identifier, which is then restricted to thirteen characters, and range from '00 to '99.

System Registers

For each student there is a set of registers. In each set there are 100 numeric registers numbered 0 to 99, and 21 string buffer registers numbered 0 to 20. The numeric registers may hold boolean, integer or real values. The string buffer registers may contain strings of up to 80 characters. The register set is local to each lesson module. They are referenced as system variables c(0) to c(99) and b$(0) to b$(20).

System Variables

All system variables are local to each lesson module. They are known only to the procedures and functions in that lesson module and to any units called from these procedures. Most of NATAL's system variables are implemented in CASTLE.

III. Implementation of the CASTLE System

Factors that are generally considered critical in the implementation of a complex system on a microcomputer are internal (RAM) and external (disk) memory limitations and the execution time of system activities. To investigate the suitability of the CASTLE design for use on a microcomputer system, a test implementation of a subset of the CASTLE language and Support System was carried out. The subset represents those aspects of the CASTLE Language and Courseware Development and Presentation Subsystems necessary to demonstrate the creation and delivery of individual lessons.

A. Selection of a System Development Language

Which language should be selected as the base language on which to build this CAI system? Since, as a minimum, it must serve the same function as the procedure language of NATAL, it should be a structured language. For this reason BASIC, the most common language on microcomputers, must be rejected. BASIC has limited structured control statements, does not have named procedures and functions, does not permit parameter passing or local variables, and variable names are significant only to the first two characters. The main advantage of BASIC is that it is easy to program and debug because of its interactive interpreter.

Pascal might be a likely candidate. It is a structured language with all the characteristics that BASIC lacks. But it has poor string handling capabilities needed in any CAI environment, and programs are more difficult to create and debug since it is a compiled language. (UCSD Pascal does have excellent string handling capabilities, but is encased in a complex environment. The Apple II version requires two manuals of over 500 pages just to explain how the system works and how this version of Pascal differs from the standard.)

The language selected as the base language for the CASTLE CAI development system is COMAL-80. Developed originally in Europe as a structured replacement for BASIC, it has attracted considerable attention on this side of the Atlantic after Commodore International placed in the public domain in May 1981 a COMAL-80 interactive editor and interpreter. COMAL (COMmon Algorithmic Language) has been referred to as the language that combines the best of BASIC with the best of Pascal. It contains all the structured control statements of Pascal; permits long name identifiers for labels, variables, and procedure and function names; allows parameter passing as call by value or call by reference; and supports local and global variables. In addition, it has excellent file handling characteristics supporting read, write and append to sequential files as well as random access files. The interactive visual editor provides for modular program development, automatic syntax checking on input, and automatic 'pretty print' listings to the screen or the printer. One of the main reasons for the selection of COMAL-80 is its string handling capabilities, superior to both BASIC and Pascal. These include selection and assignment of substrings, and a string inclusion function that returns the index of a substring in a string.

B. Hardware Selection

The decision to use COMAL-80 as the system development language for the CASTLE system, necessitated the selection of hardware that supports this language. In North America, COMAL is only available on microcomputers produced by Commodore Business Machines. CBM COMAL-80 Revision 0.12, operates on the PET 4032 and the CBM 8032 microcomputers. It only leaves 5590 bytes of memory free for programs and data. However, the CBM 8096 microcomputer runs the full extended version, CBM COMAL-80 Revision 01.02, providing a workspace of 38692 bytes. Thus, this version of COMAL and the CBM 8096 formed the basis of the development system.

The CBM 8096 is controlled by a 6502A microprocessor and contains 98304 (96K) bytes of read/write (RAM) memory and 18432

(18K) bytes of read only (ROM) memory. 14K of this ROM is occupied by Commodore BASIC 4.0. The remaining 4K is the operating system which supports I/O to a CRT screen, a keyboard, two tape drives, an IEEE-488 parallel communications bus, and a special user port. The operating system also has a built in machine language monitor.

The CRT (display) screen and the keyboard are part of the main unit. Twenty-five lines of eighty characters in two predefined fonts can be displayed on the green phosphor CRT. Each character is defined in a rectangle of 8 by 8 pixels giving a screen resolution of 640 by 200 pixels. The screen lines may be displayed in an open format, the default, which provides two blank pixel rows between each line, or closed format, with no inter-line spacing. The default font contains upper and lower case letters, numerics, 28 symbols, and 38 graphics characters. The alternate font replaces the lower case letters with 26 further graphics characters. A full business type keyboard with a separate numeric keypad is provided. It has special function keys that provide for cursor movement, screen editing, program interruption, and the reverse video display of selected characters.

The IEEE-488 bus is used to connect intelligent peripherals to the main unit. An intelligent peripheral has its own CPU, operating system, and memory, and can function independently of the 8096. Two such peripherals are needed to round out the CASTLE hardware needs: a disk unit and a printer. A Commodore 4040 disk unit was used. It has two disk drives with a storage capacity of 174848 bytes (organized into 683 blocks of 256 bytes) per drive. There may be 144 directory entries per diskette. The 4040 operating system (DOS 2.0) supports direct (random) access files which are required for the UNIT string files. Access to records in these random files is very fast. A maximum of two disk reads are required to retrieve any record.

A commodore 4022 printer was used as the printing device. It can print up to 80 characters per line with variable lines per page at different line spacing. All characters that can be displayed on the CRT can be printed including reverse video (white characters on a black background). The printer can be set in page mode so that after 60 lines are printed an automatic form feed is performed before the next line is printed. With the printer, hard copy listings of lesson modules and instructional units are easily obtained as are reports generated from student sessions.

C. Routines and Modules Developed

COMAL provided many of the general functions intrinsic to NATAL. Other NATAL functions were provided to CASTLE authors as COMAL routines. These included the general functions CNUM, EVAL, PMATCH and ROUND; the comparison functions CC (compare character), CK (compare keyletter), and CN (compare numeric); and all the NATAL edit functions. A limited set of graphic functions were also provided. Authors could also develop their own general, edit, comparison and graphic functions using the COMAL program development system. A system variable, RESPOND'CHARS$, could be set by authors to contain a list of keyboard characters that would be accepted by the UNIT RESPONSE statement. Authors create their lesson modules using the COMAL program development system.

CASTLE System modules were also developed using COMAL. These included the CASTLE initialization module, the instructional unit language interpreter, the display sub-language interpreter, the instructional unit development system, and the student mode run-time system.

D. Instructional Unit Development

The instructional unit development system has three major parts: the UNIT file manager, the UNIT editor, and the UNIT tester. The CASTLE author enters the unit development system from the COMAL program development system by selecting the UNIT file manager.

Each instructional unit is stored as two disk files. One file is a table of integers called the parse table. Each entry represents either an internal reference value, an operation code (opcode), an immediate integer value, a pointer to another part of the parse table, or a pointer to a record in the other file. The second file, called the string table, is a random access file of string records with a maximum length of 78 characters. Each record holds text data or commands for the display sub-language interpreter, or numeric expressions in string form that are evaluated at run-time. The unit is executed by the CASTLE UNIT interpreter. It carries out the actions specified by the opcodes in the parse table.

1. The UNIT File Manager:
   On entering the UNIT File Manager a list of the current units of the selected course is displayed. The following menu items may then be selected:
   a. copy a unit - A unit from an external source may be copied to the current unit directory.
   b. delete a unit - A unit may be deleted from the current unit directory and its disk files erased.
   c. edit a unit - The UNIT Editor is called for a selected unit.
   d. history of a unit - The date/time of unit creation, the date/time of the last unit update, and the unit header comment are displayed for a selected unit.
   e. list unit names - The list of unit names in the current directory is displayed.
   f. quit - Control is returned to the COMAL program development system.
   g. rename a unit - The name of a unit may be changed.
   h. start a new unit - The author registers the name of a new unit for this directory, its disk files are initialized, and a header comment by the author is placed in the unit header record.
   i. test a unit - The UNIT Tester is called for a selected unit.
2. The UNIT Editor:
   The UNIT Editor utilizes the principles of incremental compilation. That is, a unit only exists as a set of internal codes in the parse table. At run-time these codes are decoded by the UNIT Interpreter which carries out the specified actions as an instructional interaction with a student. The UNIT Editor is used by the CASTLE author to create, examine, and modify these internal codes. However, the author is never aware of the actual codes but only of their meaning as given in the UNIT Language specification.

   The use of incremental compilation has two advantages. At run-time an interpreter can execute the parse codes in the parse table much faster than if it had to parse out the meaning from UNIT language source text. Secondly, the special editor helps the author create and modify a unit by presenting options as selections from a menu and interrogating for specific parameters. At each step of the process the editor builds error-free code in the parse table.

   The UNIT Editor command level presents the author with some current statistics on the unit and a menu of options. The statistics are the amount of free space in the parse table and the number of records still available in the string table. The menu options are as follows:
   a. display statement editor - This selects a sub-editor for the DISPLAY statement. All sub-editors either interrogate the author for statement parameters or present options via menu selection. The <text> portion of the DISPLAY statement is edited by the CASTLE System Text Editor. This editor displays a window of up to 20 lines of the <text> and offers the following menu selected options:
      1) back - Move the display window back 15 lines.
      2) copy - Copy a range of lines and insert them before a specified line.
      3) delete - Delete a range of lines.
      4) edit - Enter the screen edit mode. This mode allows

the free movement of the cursor in the current display window and the changing, deletion, and insertion of characters on the display. The changes to a particular line are recorded when the RETURN key is pressed while the cursor is anywhere on that line. To exit the screen edit mode the STOP key is pressed before pressing the RETURN key. The window is redisplayed to confirm the changes.

5) first – Display the first 20 lines of <text>.
6) insert – Insert lines of text before a specified line. Any number of lines may be inserted. To exit the insert mode enter a line containing only a period (.) in the first character position.
7) last – Display the last 20 lines of <text>
8) move – Move a range of lines and insert them before a specified line.
9) next – Move the display window forward 15 lines.
10) quit – Exit the text editor.
11) search – A search is initialized for a specified string within the <text>. If the search is successful a window is displayed having as its first line the line containing the found string. The editor then enters the screen edit mode placing the cursor on the found string. On leaving screen edit mode the author may select to continue searching for the same string or exit to the text editor menu.

It should be noted that all of <text> resides in records in the unit string table on disk. These records are read, created, and updated directly on the disk by the text editor. Because of limited main memory on most microcomputers, keeping the string table on disk allows it to grow much larger than it could otherwise. This is especially important in CAI when text is used extensively.

b. exit to unit file manager – Return control to the UNIT File Manager.
c. list unit to screen – The parse table and string table are scanned by the editor to generate and display the unit's statements in human readable form.
d. print unit on printer – This performs the same function as the previous item except that the listing is sent to the printer.
e. quit – Control is returned to the COMAL program development system.
f. response statement editor – This selects a sub-editor for the RESPONSE statement.
g. test this unit – The UNIT Tester is called for this unit.
h. unit body editor – This selects a sub-editor for the statements in the unit body. The author is able to step forward and backward through the statements in the unit body. Only the statement type (EDIT, Categorization, Reprise, COMMENT, END OF UNIT) is displayed. At any step the author may select any of the following options:
   1) back – Move back one statement.
   2) delete – Delete the current statement.
   3) end – Go to the END OF UNIT statement.
   4) first – Go to the first statement.
   5) insert – Insert a statement before the current statement.
   6) list – List the current statement in full.
   7) modify – This allows the author to make modifications to the current statement.
   8) next – Move forward one statement.
   9) quit – Exit the unit body editor.

   The <text> portion of any Reprise or COMMENT statements is created and modified by a call to the CASTLE System Text Editor.

3. The UNIT Tester:

This is a special CASTLE lesson module that allows an author to execute a single unit. Before and after each test run of a unit the author may examine or change the contents of system variables, counters, and registers. After each test run the following options are presented to the author:

a. call unit file manager – Return control to the UNIT File Manager.
b. edit this unit – The UNIT Editor is called for this unit.
c. quit – Control is returned to the COMAL program development system.
d. rerun this unit – Execute the unit again.

E. The UNIT Tables and Codes

As was described above, the CASTLE UNIT exists only as internal integer codes in the unit parse table with text, display sub-language commands, and numeric expressions kept in the unit string table.

The string table always resides on disk as a random access file. It is modified directly on the disk by the UNIT Editor and the CASTLE System Text Editor. Its records are directly accessed by the UNIT Interpreter and the Display Sub-language Interpreter. In this implementation the maximum number of string table records is 3000. During editing a forward pointer, a backward pointer and a parse table reference pointer are kept in memory for each string table record. The size of this pointer table is held to 3000 records because of internal memory limitations. When records are added, inserted or deleted, these pointers are updated. At the conclusion of an editing session, the string table is re-assembled on disk from the information in the internal pointer table. The first record always contains the unit's creation date/time, its last revision date/time, and a 30 character header comment.

The parse table is stored on disk and kept in RAM memory as an integer vector with a maximum length of 1000. The length of 1000 was arbitrarily chosen in an attempt to provide adequate storage for the code of even a fairly large unit while still conserving as much memory as possible. Both the editing and execution of the parse table is carried out in main memory. COMAL provides for the transfer of entire arrays, which includes vectors, between internal RAM and disk which is very fast and straight forward.

The parse table is divided into a fixed part of 34 integers and a variable part of one or more integers. The fixed part contains counters of the number of integers in the parse table and the number of records in the string table, offsets to the code in the parse table of the first Reprise RENTRY statement and the last Reprise statement in each category, and the parameters for the DISPLAY and RESPONSE statements. The variable part contains the code for all the statements in the unit body.

F. The CASTLE Performance Recording Subsystem

When a student signs-on to a CASTLE course a performance file is opened recording the student's name, the course name, and the date and time of sign-on. When the student terminates the session, the sign-off time is recorded.

During course execution the author controls the recording of further performance records. Two types of performance records are provided by the CASTLE system. A system switch variable, PRECORDS, is set FALSE when a student enters a lesson. If the author sets PRECORDS TRUE then after each response by the student, the system automatically records the name of the unit, the student's actual response, and how the response was categorized by the unit.

Additionally the author may issue a RECORD command at any point in the lesson. This command records the contents of all system counters plus any parameters the author includes with the command. The command, RESET COUNTERS, will reset all system counters to zero. Thus the RECORD command can be used to record periodic summaries of the student's pregress in a lesson.

A CASTLE system program displays summaries of these performance records on the screen or prints them on the printer.

IV. Conclusion

A. Using the CASTLE System

During the summer of 1983, fourteen students were introduced to the COMAL/CASTLE system as part of an introductory course in computer-assisted learning at the University of Alberta. Many had no previous programming experience. As part of their laboratory assignments they wrote programs in COMAL and instructional units in CASTLE.

In a follow-on CAI project course four of these students developed and tested CAI lessons using the COMAL/CASTLE system. The lessons developed were:

1. An introduction to logic gates and truth tables
2. An introduction to current, voltage and resistance
3. A tutorial and drill in the addition of integer numbers
4. An introduction to the history of computers and computing.

The duration of lessons were from 15 to 40 minutes. All the students found the system easy to use and to adapt to their needs. The development, testing and revision cycle was facilited by the modular, interactive, and interogative design of the CASTLE system.

During student execution of CASTLE lessons, the following timings were typical:

1. the initialization of the lesson – 1.1 seconds on average
2. the initialization of a unit from the moment it is called until the first DISPLAY <text> appears on the screen – 2.5 seconds on average
3. the answer analysis response time from the moment the student enters an answer by pressing the RETURN key until the Reprise <text> appears on the screen – from under 2 seconds for simple response analyses to 7.7 seconds for the more complex response analyses.

B. Future Development

The new Commodore 64 is an inexpensive and popular microcomputer that is likely to be found in an increasing number of schools. It supports 16 colour text and graphics, a three voice music synthesizer, user defined character fonts, joysticks, paddles, and a light pen. A cartridge containing CBM COMAL-80 Revision 2.0 will soon be available for this microcomputer. This version of COMAL will include commands that emulate LOGO graphics. Packages, which allow for the grouping of procedures, functions, and data structures into special disk files which can be dynamically invoked from running programs or other packages, are also supported. This would be an ideal environment for the future development of the CASTLE System.

An economic multi-user CASTLE system of 16 Commodore 64's might be developed around Canadian Micro Distributors' MUPET II controller and ORWELL operating system, and a Commodore 9090 7.5 megabyte hard disk drive.

References

Roblyer, M.D. When is it 'good courseware'? Problems in developing standards for microcomputer courseware. *Educational Technology*, 1981, *21*(10), 47-54.

Voyce, S. *A multilingual-interpreter system for languages used in computer assisted instruction.* Toronto: The Ontario Institute for Studies in Education, 1979.